**Project Scaffold: llm-fund-matcher**

**Directory Structure**

llm-fund-matcher/

│

├── data/

│ ├── funds.csv # Provided fund dataset

│ └── queries\_sample.csv # Sample queries for testing

│

├── src/

│ ├── \_\_init\_\_.py

│ ├── data\_loader.py # Load + clean fund data

│ ├── embedder.py # SentenceTransformer wrapper

│ ├── vector\_store.py # FAISS indexing & search logic

│ ├── scorer.py # Metadata + string scoring

│ ├── matcher.py # Core match logic

│ └── config.py # Config variables & paths

│

├── app/

│ ├── cli.py # Typer-based CLI

│ ├── api.py # FastAPI endpoints

│ └── ui.py # Streamlit UI

│

├── models/

│ └── saved\_embeddings.npy # Precomputed embeddings

│

├── notebooks/

│ └── eda.ipynb # Initial exploration

│

├── tests/

│ └── test\_matcher.py # Unit tests for matcher

│

├── requirements.txt

├── README.md

└── run.py # Entrypoint for CLI

**requirements.txt**

pandas

numpy

sentence-transformers

faiss-cpu

typer

streamlit

fastapi

uvicorn

loguru

python-dotenv

fuzzywuzzy

python-Levenshtein

**src/data\_loader.py**

import pandas as pd

def load\_fund\_data(csv\_path: str) -> pd.DataFrame:

"""Load and clean fund dataset."""

df = pd.read\_csv(csv\_path)

# TODO: Add cleaning (e.g., drop duplicates, fill NaNs)

df = df.rename(columns={df.columns[0]: 'id', df.columns[1]: 'name'})

return df

**src/embedder.py**

from sentence\_transformers import SentenceTransformer

import numpy as np

class Embedder:

def \_\_init\_\_(self, model\_name: str = "sentence-transformers/all-MiniLM-L6-v2"):

self.model = SentenceTransformer(model\_name)

def encode(self, texts: list[str]) -> np.ndarray:

"""Generate embeddings for a list of texts."""

return self.model.encode(texts, convert\_to\_numpy=True)

**src/vector\_store.py**

import faiss

import numpy as np

class VectorStore:

def \_\_init\_\_(self, dim: int):

self.index = faiss.IndexFlatL2(dim)

self.ids: list[str] = []

def build(self, embeddings: np.ndarray, ids: list[str]):

"""Add embeddings and corresponding IDs to the FAISS index."""

self.index.add(embeddings)

self.ids = ids

def search(self, query\_emb: np.ndarray, top\_k: int = 10) -> list[tuple[str, float]]:

"""Return top\_k (id, distance) pairs for the query embedding."""

distances, indices = self.index.search(query\_emb, top\_k)

results = []

for dist, idx in zip(distances[0], indices[0]):

results.append((self.ids[idx], float(dist)))

return results

**src/scorer.py**

from fuzzywuzzy import fuzz

def compute\_score(query: str, title: str, metadata: dict) -> float:

"""Compute a combined score using title similarity and metadata."""

title\_score = fuzz.token\_set\_ratio(query, title) / 100

# TODO: incorporate metadata features into score

return title\_score

**src/matcher.py**

from src.embedder import Embedder

from src.vector\_store import VectorStore

from src.scorer import compute\_score

import pandas as pd

class Matcher:

def \_\_init\_\_(self, embedder: Embedder, vector\_store: VectorStore, fund\_df: pd.DataFrame):

self.embedder = embedder

self.vector\_store = vector\_store

self.fund\_df = fund\_df

def match(self, query: str) -> dict:

"""Match user query to best fund name."""

q\_emb = self.embedder.encode([query])

candidates = self.vector\_store.search(q\_emb)

best = None

best\_score = -1.0

for fund\_id, \_ in candidates:

row = self.fund\_df[self.fund\_df['id'] == fund\_id].iloc[0]

score = compute\_score(query, row['name'], row.to\_dict())

if score > best\_score:

best\_score = score

best = {'id': fund\_id, 'name': row['name'], 'score': score}

return best

**src/config.py**

import os

from dotenv import load\_dotenv

load\_dotenv()

DATA\_PATH = os.getenv('DATA\_PATH', 'data/funds.csv')

EMBEDDING\_MODEL = os.getenv('EMBEDDING\_MODEL', 'sentence-transformers/all-MiniLM-L6-v2')

EMBEDDINGS\_PATH = os.getenv('EMBEDDINGS\_PATH', 'models/saved\_embeddings.npy')

**app/cli.py**

import typer

import numpy as np

from src.data\_loader import load\_fund\_data

from src.embedder import Embedder

from src.vector\_store import VectorStore

from src.matcher import Matcher

from src.config import DATA\_PATH, EMBEDDINGS\_PATH

app = typer.Typer()

@app.command('build-index')

def build\_index():

df = load\_fund\_data(DATA\_PATH)

embedder = Embedder()

names = df['name'].tolist()

embeddings = embedder.encode(names)

np.save(EMBEDDINGS\_PATH, embeddings)

store = VectorStore(embeddings.shape[1])

store.build(embeddings, df['id'].tolist())

typer.echo('Index built successfully.')

@app.command('search')

def search(query: str):

df = load\_fund\_data(DATA\_PATH)

embeddings = np.load(EMBEDDINGS\_PATH)

store = VectorStore(embeddings.shape[1])

store.build(embeddings, df['id'].tolist())

embedder = Embedder()

matcher = Matcher(embedder, store, df)

result = matcher.match(query)

typer.echo(f"Best match: {result['name']} (score: {result['score']:.2f})")

if \_\_name\_\_ == '\_\_main\_\_':

app()

**app/api.py**

from fastapi import FastAPI

from pydantic import BaseModel

import numpy as np

from src.data\_loader import load\_fund\_data

from src.embedder import Embedder

from src.vector\_store import VectorStore

from src.matcher import Matcher

from src.config import DATA\_PATH, EMBEDDINGS\_PATH

class Query(BaseModel):

query: str

app = FastAPI()

# Load resources on startup

df = load\_fund\_data(DATA\_PATH)

embeddings = np.load(EMBEDDINGS\_PATH)

store = VectorStore(embeddings.shape[1])

store.build(embeddings, df['id'].tolist())

embedder = Embedder()

matcher = Matcher(embedder, store, df)

@app.post('/search')

def search\_fund(q: Query):

result = matcher.match(q.query)

return result

**app/ui.py**

import streamlit as st

import numpy as np

from src.data\_loader import load\_fund\_data

from src.embedder import Embedder

from src.vector\_store import VectorStore

from src.matcher import Matcher

from src.config import DATA\_PATH, EMBEDDINGS\_PATH

st.title('LLM Find My Fund')

df = load\_fund\_data(DATA\_PATH)

embeddings = np.load(EMBEDDINGS\_PATH)

store = VectorStore(embeddings.shape[1])

store.build(embeddings, df['id'].tolist())

embedder = Embedder()

matcher = Matcher(embedder, store, df)

query = st.text\_input('Enter your fund query')

if st.button('Search'):

result = matcher.match(query)

st.write('\*\*Best match:\*\*', result['name'])

st.write('\*\*Score:\*\*', f"{result['score']:.2f}")

**run.py**

import typer

from app.cli import app as cli\_app

if \_\_name\_\_ == '\_\_main\_\_':

cli\_app()

**README.md**

# LLM Find My Fund

## Setup

```bash

pip install -r requirements.txt

**Build Index**

python run.py build-index

**Search via CLI**

python run.py search "SBI tech growth"

**Run API**

uvicorn app.api:app --reload

**Run UI**

streamlit run app/ui.py

---

## tests/test\_matcher.py

```python

import pytest

import numpy as np

from src.data\_loader import load\_fund\_data

from src.embedder import Embedder

from src.vector\_store import VectorStore

from src.matcher import Matcher

def test\_match\_exact():

df = load\_fund\_data('data/funds.csv')

embedder = Embedder()

embeddings = embedder.encode(df['name'].tolist())

store = VectorStore(embeddings.shape[1])

store.build(embeddings, df['id'].tolist())

matcher = Matcher(embedder, store, df)

result = matcher.match(df['name'].iloc[0])

assert result['name'] == df['name'].iloc[0]